Math

UIUCDCS-R-80-1025

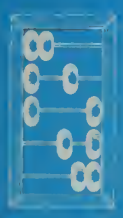
UILU-ENG 80 1717

APPLICATION OF THE FAULT-TOLERANT DEADLINE MECHANISM TO A SATELLITE ON-BOARD COMPUTER SYSTEM

by

A. Wei, K. Hiraishi
R. Cheng, R. Campbell

June 1980



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-80-1025

APPLICATION OF THE FAULT-TOLERANT DEADLINE MECHANISM TO A SATELLITE ON-BOARD COMPUTER SYSTEM

by

A. Wei, K. Hiraishi
R. Cheng, R. Campbell

June 1980

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

Supported by NASA NSG 1471.

1 Introduction.

A real-time system provides a set of services subject to timing constraints. The design of such system is often expensive and error prone because of concurrency and timing constraints. Moreover, many highly reliable applications require the use of real-time computer systems; such as fly-by-wire passenger aircraft, nuclear reactor safety systems, etc. While several approaches to reliable hardware exist, there are few methodologies that permit the construction of reliable software. Fault-tolerant software is required to prevent system failure caused by accident, sabotage, or residual hardware/software design faults (that is, faults not detected by proof, inspection, or testing). This paper discusses fault-tolerance in high level programming languages for real-time systems and reviews the motivation and design of the deadline mechanism [1]. An application of the mechanism to a satellite on-board computer system is described which demonstrates the potential of the deadline mechanism for programming fault-tolerant real-time processes.

The definitions of concepts involved in real-time fault-tolerance are based on the paper [1]. It is assumed that timing constraints are imposed upon a system by its specification. Interactions between the system and an environment are performed by service components of the system. These service components receive requests from the environment, perform a service, and then respond to the environment.

A system clock is assumed to be an essential component of a real-time system. System time is part of the internal state of the real-time system. It is assumed that consistent views of the times in external environments can

be obtained as functions of the system clock.

Associated with each service component is a request time (the value of the system clock when a request is detected) and a response time (the value of the system clock by which a response must be completed). A response period is the maximum allowable amount of system time that can elapse between a request time for a particular service and the corresponding response time for that service. The response period is part of the specification of the system. A request period is the interarrival time of requests for the service measured with respect to the system clock and is also part of the specification. A deadline is the system time by which a system must respond to a request and is the sum of the request time and response period for a given service component. An execution period is the maximum amount of system time required to execute a block of instructions assuming no residual faults and uninterruptability. This is a measurable quantity and is determined for particular blocks of instructions executing on specific processors. The execution period of a service component is called a service period.

2 Problems Associated with Real-Time Systems.

2.1 Reliability Issues in Real-Time Systems.

In real-time systems, software is a vital component whose failure may result in catastrophe. For example, in defense missile systems the possibility of software failure can make the whole system unacceptable. In an aircraft control system a failure may lose human lives, and in nuclear reactor safety

systems a failure may result in a disaster. Hence the requirement that real-time systems meet their specifications is crucial.

Reliable software has been traditionally obtained through fault-avoidance techniques. Minimizing system complexity, top-down modular design, structured programming, and proving program correctness are all efforts to achieve this goal. However, a fault-tolerance approach [2] is required because of at least the following reasons:

1) It is always impossible to guarantee the absence of faults,
2) The delays and interruptions of real-time systems (air traffic control, process control, etc.) caused by manual repair actions is unacceptable,
3) Manual intervention is inaccessible for some systems (space, undersea, etc.), and
4) High cost of lost time and of manual maintenance is unacceptable in many installations.

One critical requirement associated with real-time system reliability is that the timing constraints are met.

2.2 Timing Faults in Real-Time Systems.

A real-time system that can always respond to a request before the deadline is characterized as timely. A timing failure occurs when a real-time system violates one of the timing constraints in its specification. A timing fault causes a timing error, and a timing error is defined as information representing an incorrect timing relationship in the internal state of the system. Timing faults may exist because of the following situations:

1) miscalculation of the execution periods of some algorithms,

2) transient hardware faults,

3) unbounded algorithms due to loops or recursions,

4) system overload,

5) resource contention, either of software resources or hardware resources , and

6) system deadlock.

Timing faults may be included in software despite control over the system specification, design, and implementation. Timing errors may also occur because of hardware or some external system failure not documented in the requirement. In order to cope with unexpected timing errors, we need an error recovery mechanism.

2.3 Problems with Assembly Languages.

Many real-time systems are implemented in assembly language. Some of the reasons for using assembly language are that the interrupts can be controlled explicitly with assembly language, machine-dependent features are easier to program, and the execution time is easy to estimate and usually smaller than that of using high level language. However, the proof of correctness of an assembly language program is difficult and the specification of timing constraints in assembly language programs is almost impossible. Simulation, repeated testing, and program redundancy can be used to improve system reliability. However, simulations may be lengthy and it is frequently difficult to simulate the system closely. Lack of time may prevent repeated testing from attempting all the possible contingencies and input values. The provision of redundancy in assembly language programs is ad hoc. Furthermore, these methods do not guarantee the timing constraints will be met. Finally,

systems developed in assembly language are usually very expensive.

2.4 Problems with High Level Languages.

With advancing technology in high speed computer hardware and increasing complexity of software systems, the current trend is to develop high level programming languages to support real-time systems [3]. HAL/S [4] developed by Intermetric Inc. has been used to program the Space Shuttle control system. Modula [5], Concurrent Pascal [6], ILIAD [7], Ada [8] and Path Pascal [9, 10] are all efforts in this direction. The advantage of high level system programming languages is obvious; it facilitates programming, understanding, modification, and most importantly, maintenance of the system. Other advantages are portability and reliability.

Reliability is greatly enhanced by the use of high level languages for real-time systems because program verification is simplified. Timing constraints are an important aspect of reliability in real-time systems and should be rigorously specified. Verification that these constraints are satisfied is, however, very difficult. It is important to provide a systematic approach to fault-tolerance in highly reliable real-time systems. Most existing high level languages, however, do not specify timing constraints. Furthermore, there is no generalized methodology in existing high level languages to tolerate timing faults or to recover from timing errors.

3 Fault-Tolerance in High Level Languages.

3.1 Fault-Tolerance in Real-Time Systems.

3.1.1 Recovery Block Scheme.

Fault-tolerant design techniques have been used for a long time to improve hardware reliability, but only recently have they been proposed to achieve high software reliability [11, 12, 13]. The recovery block scheme [11, 14] has been proposed as a mechanism to support fault-tolerant software. A simple recovery block is shown below.

```
ensure  acceptance_test
by      primary
else by alternate
else error;
```

The acceptance_test is an assertion about the results computed by the primary or the alternate. If the acceptance test is false after a primary execution, the alternate is executed. Errors detected implicitly by the hardware or software during the execution of the primary or the alternate automatically set the acceptance test to false. If both the primary and alternate of a recovery block fail to satisfy the acceptance test, a software error is generated. The alternate performs the same function as the primary but may use a different algorithm. A hardware device called the recovery cache [15] is used to restore the state after a primary failure to the state prior to entry to the recovery block. The recovery block scheme has been implemented in Sequential Pascal [16] and hardware caches have been constructed for the PDP-11 [17]. The recovery block scheme is applied by Hecht to real-time applications [18]. Hecht augments the acceptance test with a watchdog timer that monitors the execution within a specified response period. Problems arise, however, if

recovery blocks are applied to software systems whose services must meet deadlines. The deadline mechanism [1], inspired by the recovery block scheme, is proposed as a means to systematically construct fault-tolerant real-time systems with guaranteed response time.

3.1.2 Deadline Mechanism.

The deadline mechanism, although similar to the recovery block scheme, performs a different function. Recovery blocks are insensitive to the passage of time and cannot be used to ensure that a response is generated by a specific deadline. (That is, it is impossible to recover from a missed deadline by resetting the system clock and executing an alternate.) Recovery blocks do provide alternate algorithms to cope with general software faults involving logical errors but can not handle timing errors properly. The deadline mechanism includes scheduling mechanisms to ensure timely responses. The recovery block and deadline mechanism complement each other and may be nested to provide tolerance to both timing and other faults.

The deadline mechanism associates two algorithms with each particular service component. The "primary" algorithm produces a better quality service than the "alternate". The alternate is a simpler, deterministic algorithm which always produces an acceptable result in a known, fixed length of time. The deadline mechanism provides fault-tolerance in a real-time system by ensuring that either the primary or the alternate will complete before the deadline. A set of simulations demonstrating the feasibility of fault-tolerance in real-time systems is described in [1].

Reliable scheduling of primaries or alternates to meet real-time constraints requires the calculation of the execution period of each algorithm.

Accurate determination of execution periods is critical to system performance and reliability. Anderson and Witty [19] proposed placing upper bounds on iterative and recursive program codes. For many desirable algorithms such upper bounds may not be easily established. The "service period" of a component can then be set, for scheduling purposes, at a figure that is less than the response period, but no less than the execution time of the alternate. Since upper bounds on the execution periods for the alternate algorithms are assumed, the deadline mechanism has been designed so that the scheduler reserves a time for execution of the alternate as requests arrive. Primaries are scheduled in any remaining time which is called slack time.

3.2 Experimental Language Construct for the Deadline Mechanism.

The deadline mechanism is a tool for specifying timing constraints explicitly in the program and provides tolerance for timing faults. We have incorporated an experimental language construct for the fault-tolerant deadline mechanism into an existing programming language, Path Pascal [9, 10]. Path Pascal was chosen as the vehicle for this experiment because of its availability and high level synchronization mechanisms. Path Pascal includes concurrent processes, device programming features, a real-time clock, an encapsulation mechanism called an "object", and Open Path Expressions [20] for synchronization of processes. Each object consists of a declaration of the data structure together with a set of operations which access the values of that structure. Objects permit synchronization to be associated with shared data and are implemented as an extension of the Pascal structure type facility. Operations accessible outside of the object are specified by prefixing the declaration of an operation with the reserved word "entry". "Interrupt

processes" permit device programming by means of a "doio" statement similar to that in Modula [5]. The real-time programming features include three built-in routines "await", "delay", and "time". Await suspends a process until the argument of await corresponds to the system time. Delay suspends a process for the number of time units specified in its argument. Time is a function which returns the current value of the system clock.

Path Pascal has been extended to include the fault-tolerant deadline mechanism. The extensions should not be considered as a language design, rather as experimental features which have allowed the programming of example reliable real-time systems. A "deadline process" provides a fault-tolerant deadline service. The response period to a request is a parameter of the deadline process. The deadline process contains one "request" and one "service" statement. The request period is specified following an "every" phrase in the "request" statement. The service statement specifies the primary and alternate algorithms. A simple example of a deadline process is shown below:

```
deadline process position_update [response = onesecond];
begin
    every fiveseconds request
        get_data_from_sensors;
        service nextposition.compute
        else nextposition.approximate;
    until navigation_terminated;
end;
```

where "nextposition" is an object containing the information of positions to be updated. "Compute" and "approximate" are two entry procedures in the object to do the position update. "Compute" is a primary algorithm and "approximate" an alternate algorithm.

In the example above, deadline process "position_update" computes the "nextposition" every five seconds starting from the instantiation of the dead-

line process. The request statement reads data from sensors and the service statement calculates the next position. Assuming that the execution period of reading sensor data and the approximating algorithm is less than the response period, the deadline mechanism ensures each request will complete before its deadline.

The rate-monotonic priority assignment [21] is used to schedule alternates using the response period as a static priority, with small values having high priority. At any given instant the alternate with the highest priority (smallest response period) executes. This scheme requires preemption of alternates. It has been shown that such a system will be timely if processor utilization is less than ln 2 (approx. 0.69) [21]. One advantage of this method is a reduction in scheduling complexity and overhead, an aspect not considered in [1]. The primaries are scheduled in earliest-deadline-first order.

Executions of the primary and alternate within the service period are ordered alternate first (the first-chance scheduling algorithm [1]). If the primary completes within its execution period, its results are used in preference to those of the alternate. If the primary should fail to complete within its execution period, the results from the alternate are used. A modified recovery cache has been implemented in a software interpreter for the language and is used to hold results from the alternate until the primary either fails or completes successfully.

4 Application to a Satellite On-board System.

A satellite on-board system has been selected as an example of a real-time system. The on-board system requires high reliable hardware and software to achieve a successful mission. Software failures, including timing failures, can degrade the system performance and inhibit the success of the mission. In the worst case, it can cause the loss of the satellite costing tens of millions of dollars. We show how the fault-tolerant deadline mechanism can be applied to the on-board system to tolerate timing faults.

4.1 The Model of the On-board System.

The model of the on-board system is based on Multimission Modular Spacecraft [22] which is the standard satellite for a wide variety of earth orbiting missions in the 1980's. The spacecraft is a zero-momentum, three axis stabilized satellite. The executive program for its on-board system has been written in a low level language [23].

A block diagram of the model is depicted in Fig. 1.

The Central Unit (CU) interfaces with the On-Board Computer (OBC), Remote Interface Units [RIUs], Command Demodulator-Decoder and the PCM Modulator. From the CU, commands are channeled through the Multiplex Data Bus (MDB) to the RIUs and from there to the various subsystems. Telemetry is sampled in the subsystems, selected by the RIUs, and then passed to the format generator in the CU through the MDB. From the CU, PCM goes to the Modulator. Both commands and PCM are channeled to the OBC. The OBC can communicate with all satellite subsystems through the CU, MDB and RIUs.

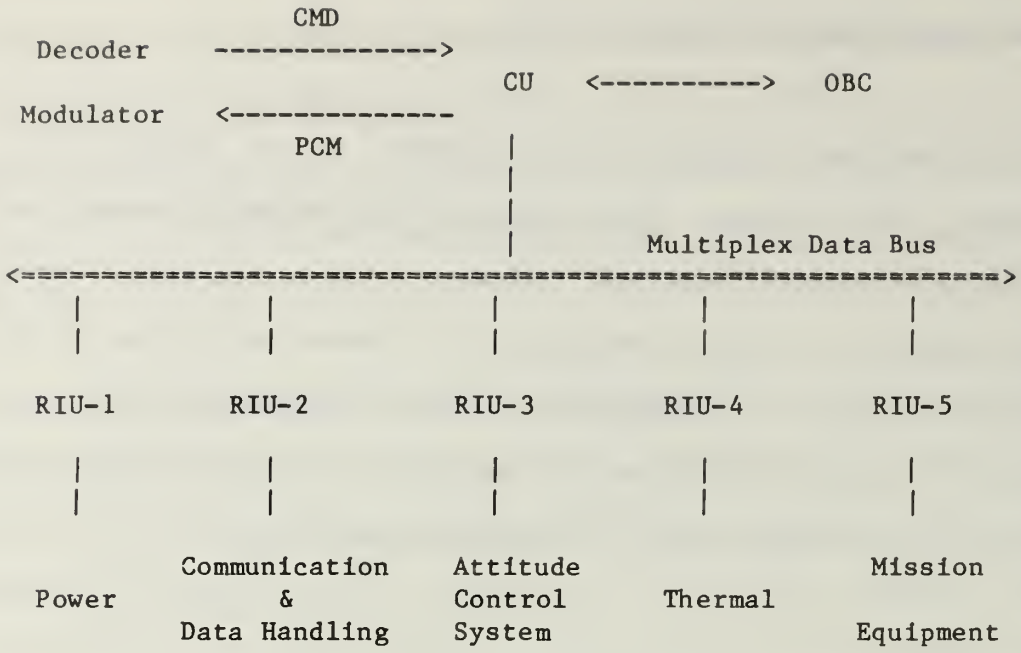


Fig. 1 Satellite On-Board System Model

Though the functions of the OBC generally depend on the mission, the attitude control, delay command process, statistical calculation and limit checking, power management, thermal control, and mission equipment management are functions commonly required by various missions.

4.2 Time Critical Subsystems.

Among the functions mentioned above, attitude control is the most complicated and the most time critical, especially during the initial acquisition phase when a big attitude maneuver is required. If the satellite is in an incorrect attitude for a long time hardware damage may result from abnormal thermal conditions. Furthermore, the solar panel can not provide power and the battery will be drained. The satellite can not survive without battery power. The satellite has very little momentum to stabilize its attitude and is very sensitive to disturbance torque. The control loop must continuously

monitor disturbance torque and provide a cancelling control torque when necessary. The big attitude maneuver requires a large control torque, and control loop failures during this period may cause the satellite to tumble. If the satellite tumbles, the radio link between the satellite and the ground stations may be lost. (The satellite antenna beam must be aligned with the ground stations.) Should this happen, the ground stations will be unable to transmit suitable recovery commands to the satellite, the satellite will become out of control and the mission will fail.

The attitude control subsystem consists of attitude sensors (gyros, sun sensor, and earth sensor), OBC, and actuators (inertial wheels, and gas jets). Gyro signals are read every 125 ms and pre-processed every half second. Sun and earth sensor signals are read every second and sent to the gyro calibration process. The gyro calibration process is executed every two seconds to provide the calibration data to the attitude determination process. The attitude determination process is the most essential and time consuming process within this subsystem. This procedure is executed every second. Actuator control signals are provided by the control process every second and sent to the inertial wheels every 125 ms.

The data flow between the processes is shown in the following timing chart (Fig. 2).

4.3 Application of the Deadline Mechanism to the Attitude Control System.

The deadline mechanism has been applied to each process of the attitude control system. For example, in the attitude determination process, a linearized Kalman filter is used as the primary procedure. The filter is

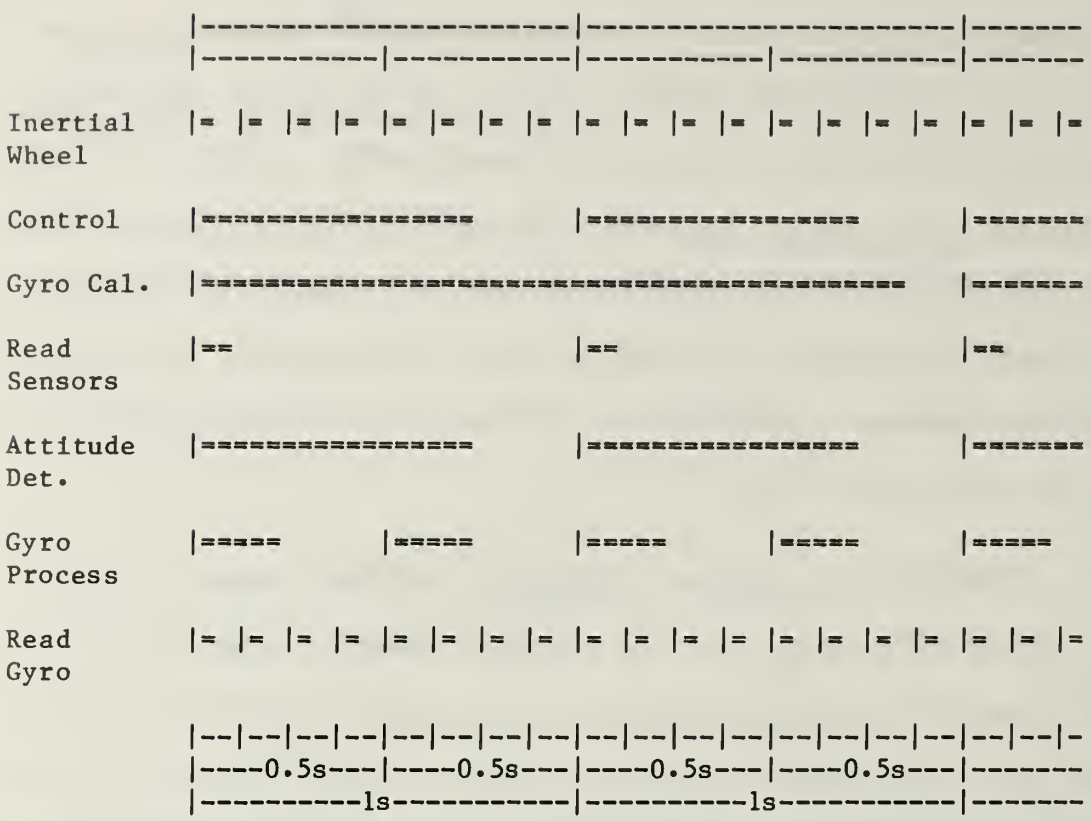


Fig. 2 Attitude Control Timing Chart

designed to determine attitude angles of the satellite and their changing rate. To achieve required accuracy, the primary becomes rather complicated. The alternate is a simple prediction algorithm by a transition matrix. This prediction algorithm is accurate enough when the time duration of the prediction is no longer than 4 or 5 seconds.

A program example for attitude determination is listed below. The example shows the structure of the program.

```
deadline process attitude_determine [response = onesecond];
local declarations

procedure filter;
begin
    kalman filter;
end;
```

```
procedure prediction;
begin
    prediction by transition matrix;
end;

begin (* attitude_determine *)
    initializations of local data;
    await(specified_time);
    every onesecond request
        service filter else prediction;
    until terminated;
end; (* attitude_determine *)
```

All the other functions mentioned in the previous section are similarly programmed as deadline processes. The synchronization among deadline processes is achieved by the "await" function.

4.4 Reliability Improvement by the Deadline Mechanism.

The deadline mechanism permits the construction of highly reliable systems from primaries that are orders of magnitude less reliable than the requirements. The following argument is based on several simplifying assumptions, however, we believe that it demonstrates the potential of the mechanism for programming reliable systems. Let the timing failure probability of the alternate and primary for a unit of time be $P[f_a]$ and $P[f_p]$, respectively. We assume that each f_a and f_p are independent and that the deadline mechanism implementation is reliable. A timing failure of an alternate causes a subsystem failure. Since the alternate is much simpler than the primary, the results calculated by the alternate includes some tolerable inaccuracy. Consecutive primary failures, although tolerated by timely execution of the alternate, accumulate the inaccuracy and produce unacceptable results. Therefore, m consecutive primary timing failures are considered to cause a subsystem failure. "m" is determined by the quality of the alternate algorithm and the system

requirements. Then the probability of the subsystem timing failure for a unit of time, denoted by $P[f_s]$, is as follows:

$$P[f_s] = P[f_a] + P[s_a]*P[mf_p] \text{ ----------(1)}$$

Where $P[mf_p]$ is the probability of m consecutive primary failures for a unit of time and $P[s_a]$ is the success probability of the alternate for a unit of time.

Since $P[mf_p] \leq \{P[f_p]\}^m$, we can rewrite (1) as:

$$P[f_a] \leq P[f_s] \leq P[f_a] + P[s_a]*\{P[f_p]\}^m \text{ ----------(2)}$$

The satellite acquisition maneuver is often designed to tolerate 99.73% of errors occurring in each of the maneuver parameters, such as control torque, disturbance torque, etc., during the maneuver. (The value 99.73% corresponds to an occurrence probability within 3 standard deviations on either side of the nominal under the assumption of normally distributed errors.) Considering the timing constraint of a deadline process as a maneuver parameter, the probability of a timing failure of a deadline process must be less than 0.0027 for the duration of the maneuver. We further assume that the duration of the maneuver is two days. For the attitude determination process, the required timing failure probability for the request period (1 second) is less than $1.56*10^{-8}$. Here we choose the request period of the attitude determination process as the unit of time and the following discussions on probability are based on this unit of time.

Without the benefit of the deadline mechanism, the timing failure probability of the primary, $P[f_p]$, must be less than $1.56*10^{-8}$. To estimate the effectiveness of the deadline mechanism, we will make the following simplifi-

cations. We can simplify equation (2) by replacing $P[s_a]$ by 1 (we assume that $P[f_a]$ is very small.) Suppose that only two consecutive failures of the primary can be allowed. Using the required timing failure probability of the subsystem, we can estimate the required timing failure probabilities of the primaries and alternates that will guarantee the reliability of the subsystem from:

$$P[f_a] < P[f_s] < P[f_a] + \{P[f_p]\}^2 < 1.56*10^{-8}$$

The alternate is much simpler than the primary and we assume that it can be constructed with a timing failure probability less than that of the subsystem (in the order of 10^{-8}). The required timing failure probability of the primary can be four order of magnitude greater than that of the alternate or the subsystem.

5 Conclusions.

Path Pascal has been extended to include the fault-tolerant deadline mechanism. The satellite on-board system has been simulated using the extended Path Pascal. The deadline mechanism provides a systematic recovery scheme for timing errors. The language has shown its potential to describe real-time systems.

The required manpower to develop the simulation program in extended Path Pascal was about six man-months. The development of the executive program by the OAO project using low level language required forty-eight man-months [23]. Though the simulation program is a simplified version of the

executive, we believe that a tremendous coding and testing time can be saved by the high level programming language approach.

Fault-tolerance in software is needed in addition to the fault-avoidance approach. Indeed, tolerance and avoidance are complementary approaches to the fault problems. We believe that theoretically and practically the reliability of a real-time system is considerably improved by applying the deadline mechanism. Even though overhead is incurred for error recovery and executing additional algorithms [24], the mechanism achieves high reliability for time critical components of real-time systems.

6 Acknowledgements.

We would like to thank NASA (Grant NSG 1471) for financing this research. Special thanks also go to Nippon Electric Co., Ltd. for providing valuable information about on-board data handling systems.

REFERENCES

[1] Campbell R. H., K. H. Horton and G. G. Belford, "Simulations of a Fault-Tolerant Deadline Mechanism," Proc. of the 9th International Conf. on Fault-Tolerant Computing, Madison, Wisconsin, June, 1979.

[2] Avizienis, A., "Fault-Tolerant Computing - Progress, Problems, and Prospects," Information Processing 77, North-Holland Pub. Co., Amsterdam, pp. 405-420, 1977.

[3] Wirth, N., "Toward a Discipline of Real-Time Programming," CACM, Vol. 20, No. 8, pp. 577-583, August, 1977.

[4] Intermetric Inc., HAL/S Manual, 1975.

[5] Wirth N., "Modula: a Language for Modular Multiprogramming," Software-Practice and Experience, 7, pp. 3-84, 1977.

[6] Brinch Hansen, P., The Architecture of Concurrent Programs, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

[7] Schutz, H. A., "On the Design of a Language for Programming Real-Time Concurrent Processes," IEEE Trans. on Software Engineering, Vol. SE-5, No. 3, pp. 248-255, May 1979.

[8] Wegner, P., "Programming with ADA: An Introduction by Means of Graduated Examples," SIGPLAN NOTICES, Vol. 14, No. 12, pp. 2-46, Dec. 1979.

[9] Campbell R. H. and R. B. Kolstad, "Path Expressions in Pascal," Fourth International Conference on Software Engineering, Munich, Germany, Sept. 1979.

[10] Campbell, R. H. and R. B. Kolstad, "Practical Applications of Path Pascal to Systems Programming," ACM79, Detroit, 1979.

[11] Randell B., "System Structure for Software Fault Tolerance," IEEE Trans. on Software Engineering, Vol. SE-1, No. 2, 1975, pp. 220-232.

[12] Denning, P. J., "Fault Tolerant Operating Systems," Computing Surveys, Vol. 8, No. 4, December, 1976, pp. 359-389.

[13] Hecht, H., "Fault-Tolerant Software," IEEE Trans. on Reliability, Vol. R-28, No. 3, August, 1979, pp. 227-232.

[14] Horning, J. J., H. C. Lauer, P. M. Melliar-Smith and B. Randell, "A Program Structure for Error Detection and Recovery," in Proc. Conf. Operating Systems; Theoretical and Practical Aspects. IRIA, 1974, pp. 177-193. (Reprinted in Lecture Notes in Computer Science, Vol. 16, Springer-Verlag, New York.)

[15] Anderson, T., and R. Kerr, "Recovery Blocks in Action: A system supporting high reliability," 2nd International Conference on Software Engineering, pp. 447-457, October, 1976.

[16] Shrivastava, S. K., "Sequential Pascal with Recovery Blocks," Software - Practice and Experience, Vol. 8, pp. 177-185, 1978.

[17] Lee, P. A., N. Ghani and K. Heron, "A Recovery Cache for the PDP-11," Proc. of the 9th International Conf. on Fault-Tolerant Computing, Madison, Wisconsin, June, 1979.

[18] Hecht H., "Fault-Tolerant Software for Real-Time Applications," Computing Surveys, Vol. 8, No. 4, 1976, pp. 391-407.

[19] Anderson, T. and R. W. Witty, "Safe Programming," BIT 18, pp. 1-8, 1978.

[20] Campbell R. H., "Path Expressions: A Technique for Specifying Process Synchronization," Ph. D. Thesis, The University of Newcastle Upon Tyne, August, 1976.

[21] Liu C. L. and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," JACM, Vol. 20, No. 1, 1973, pp. 46-61.

[22] NASA Godard Space Flight Center, "Multimission Modular Spacecraft System Specification," S-700-10, May 1977.

[23] Taylor T., "Modular Multi-Mission Spacecraft," Presentation Material for ICASE/NASA Langley Workshop on Real-Time Programming for NASA Flight Projects, October 1979.

[24] Liestman, A. L. and R. H. Campbell, "A Fault-tolerant Scheduling Problem," Tech. Report UIUCDCS-R-80-1010, Dept. of Computer Science, University of Illinois, Feb. 1980.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-80-1025	2.	3. Recipient's Accession No.

4. Title and Subtitle

Application of the Fault-Tolerant Deadline Mechanism to a Satellite On-Board Computer System

5. Report Date: June 1980

6.

7. Author(s): A. Wei, K. Hiraishi, R. Cheng, R. Campbell

8. Performing Organization Rept. No. R-80-1025

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois U-C
Urbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.: NASA NSG 1471

12. Sponsoring Organization Name and Address

NASA Langley Research Center
Hampton, Virginia 23665

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

Many real-time systems require high reliability both in software and hardware. While several approaches to reliable hardware components exist, there are few methodologies that permit the construction of reliable software. Fault-tolerant software is required to prevent system failure caused by accident, sabotage, or residual hardware/software design faults (that is, faults not detected by proof, inspection, or resting). A deadline mechanism based on recovery blocks is briefly described which permits recovery from timing faults. A language construct for the deadline mechanism is incorporated into the Path Pascal programming language. Application of the fault-tolerant deadline mechanism to a satellite on-board computer system is described using extended Path Pascal as a vehicle. The MMS (Multimission Modular Spacecraft), a satellite designed by NASA for the 1980's, is used as model.

17. Key Words and Document Analysis. 17a. Descriptors

Path Pascal
fault-tolerant
deadline mechanism

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement: unlimited

19. Security Class (This Report): UNCLASSIFIED

20. Security Class (This Page): UNCLASSIFIED

21. No. of Pages: 23

22. Price

FORM NTIS-35 (10-70)

USCOMM-DC 40329-P71